APPLICATION FOR UNITED STATES LETTERS PATENT

FOR

Configuration Proxy Service for the Extended Firmware Interface Environment

First Named Inventor: Russell Carr

Prepared by: Blakely, Sokoloff, Taylor & Zafman LLP
12400 Wilshire Boulevard, 7th Floor
Los Angeles, California 90025
(408) 720-8300

Attorney Docket No. 42390P12030

"Express Mail" mailing label number EL867647941US

Date of Deposit January 15, 2002

I hereby certify that this paper or fee is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 CFR 1.10 on the date indicated above and is addressed to the Commissioner of Patents and Trademarks, Washington, D.C. 20231

Christopher P. Marshall
(Typed or printed name of person mailing paper or fee)

C.P. Marshall
(Signature of person mailing paper or fee)

1/15/02
(Date)

Configuration Proxy Service for the Extended Firmware Interface Environment

FIELD OF THE INVENTION

[0001] The present invention pertains to the field of field diagnostics support. More particularly, the present invention relates to a method to provide auto-configuration of diagnostic test models in the field to match the system's configuration.

BACKGROUND OF THE INVENTION

[0002] Firmware is a set of hardware setup programs. Firmware is typically stored in flash memory or read only memory. Firmware includes programs such as device drivers, which are used to activate the hardware of a standard computer system. For example, a personal computer may include hardware for performing operations such as reading keystrokes from a keyboard, transmitting information to a video display, or sending information to a printer.

[0003] The operating system (OS) performs functions such as scheduling application programs and resolving conflicts among applications that request access to the same resources. Moreover, operating systems communicate service requests from application programs to the hardware device drivers. Examples of operating systems include DOS, Windows, and UNIX.

[0004] The Extensible Firmware Interface (EFI) is an architecture specification. The EFI provides an interface between the OS and the platform firmware. The interface is in the form of data tables that contain platform-related information, and boot and runtime service calls that are available to the OS and

its loader. The EFI helps provide a standard environment for booting the OS and running system maintenance applications.

[0005] The EFI provides a coherent, scalable platform environment. The specification defines a complete solution for the firmware to completely describe platform features and surface platform capabilities to the OS during the boot process. The EFI definitions are compatible with both 32 bit Intel Architecture (IA-32) and 64 bit Intel Architecture (IA-64) based processors.

[0006] Figure 1 shows the principal components of EFI and their relationship to the platform hardware 150 and the OS software 110. The platform firmware is able to retrieve the OS loader image from the EFI system partition 100. The specification provides for a variety of mass storage device types including disk, CD-ROM, and DVD as well as remote boot via a network.

[0007] Once the platform firmware has begun to retrieve the OS loader image, the OS loader 120 continues to boot the complete operating system 110 at approximately the same time. In doing so, the OS loader may use the EFI boot services 130 and interfaces defined by EFI or other required specifications 160 to survey, comprehend, and initialize the various platform components and the OS software that manages them. The EFI runtime services 140 and console services 170 are also available to the OS loader 120 during the boot phase.

BRIEF DESCRIPTION OF THE DRAWINGS

[0008] The present invention is illustrated by way of example and not limitation in the figures of the accompanying drawings, in which like references indicate similar elements and in which:

[0009] FIG. 1 shows a block diagram of the Extended Firmware Interface;

[0010] FIG. 2 shows a flowchart of an embodiment of the configuration proxy abstracting configuration information prior to booting the system;

[0011] FIG. 3 shows a block diagram that illustrates an embodiment of a platform diagnostic architecture;

[0012] FIG. 4 shows a block diagram of an embodiment of the configuration proxy.

DETAILED DESCRIPTION

[0013] The configuration proxy service provides configuration information for the interface between the OS and the platform firmware. For one embodiment of the invention, the interface is the EFI service environment. The configuration information is abstracted from the platform's underlining configuration reporting standards prior to booting the system. The machine is booted to the EFI service partition prior to running the OS. Once the operating system begins running, the OS activity is given priority over the EFI service environment and the configuration proxy operation is terminated.

[0014] Figure 2 provides an example of one embodiment of the configuration proxy abstracting configuration information prior to booting the system. After startup or reset of the machine in operation 200, the EFI service environment is booted in operation 210. A list of available devices to test is retrieved in operation 220. The configuration proxy, in operation 230, enumerates the system configuration information. The system configuration information is then stored in memory in operation 240.

[0015] The configuration proxy is designed to provide a high degree of isolation between the test algorithms and the details of a platform's implementation. Thus, the configuration proxy is portable across different platforms and execution environments. The portability of the configuration proxy will help reduce maintenance and development costs of test modules. Test programmers will not need to write and maintain their own discovery algorithms.

Instead of designing test algorithms specific to a given platform, the configuration proxy allows test algorithms to be used with multiple platforms by obtaining necessary configuration information through object class definitions. This removes dependence on independently developed and maintained methodologies. The underlying discovery infrastructure will be allowed to evolve independently of the tests in support of new architectures and features.

[0016] The discovered or saved information can be displayed by a local console or by remote applications that use the data. The discovered or saved information can also be used to auto-configure a test suite. Further, the information can be assessed by tests directly, as in operation 250 of Figure 2.

[0017] Aside from diagnostics, the configuration proxy can be used with other applications such as utilities. The applications are used locally within the EFI service environment or remotely through protocols through the service environment. In the event that the applications are remote, the applications would be accessed through an EFI driver stack via either a local area network or a serial bus. If remote, the applications may be hosted by any operating system.

[0018] As previously stated, once the OS begins running in operation 260, the OS activity is given priority over the EFI service environment and the configuration proxy operation is terminated in operation 270.

[0019] Figure 3 depicts the high level architecture of a system, which includes a configuration proxy. The diagnostic console 300 sends a request to test control 310 to retrieve the list of available devices to test. The

communication between the diagnostic console 300 and the test control 310 ensures that the tests available on the system are properly associated with the devices available. The test control 310 provides the system with the ability to test the hardware that the operating system is going to be used with. For one embodiment of the invention, a predetermined "gold configuration" is compared with the discovered configuration. If a bug is found in the firmware or the system did not discover the firmware either because the device is not responding or the firmware has a bug, the test control 310 issues a flag.

[0020] The test control module 310 utilizes the configuration proxy 330 to enumerate the system configuration information from the platform's firmware tables 390. The configuration information is retrieved either in object format as retrieved from the firmware tables 390 or plain text form. The configuration proxy 330 is not designed to be modified by users. For an embodiment of the invention, the platform firmware information obtained by the configuration proxy 330 is generated by whatever firmware tables are available. All abstracted information is made available and the tester is free to use whatever information is applicable.

[0021] For another embodiment of the invention, the test control 310 has knowledge of exactly which devices exist. In this embodiment, the test control 310 discovers which devices are available, gathers only information needed by the diagnostic tests, and applies the tests using the information gathered. There

is a possibility that tests which access the test control 310 will require more information.

[0022] For yet another embodiment of the invention, the tests are able to access the configuration proxy 330 directly without having to access the test control 310. Tests may use the information from the configuration proxy 330 to access hardware through the drivers or port addresses which were configured by the EFI firmware, rather than having to have apriority knowledge of the system's configuration.

[0023] The platform's firmware tables 390 may include System Management Basic Input/Output System (SMBIOS) tables, Advanced Configuration and Power Interface (ACPI) tables, and Intelligent Platform Management Interface (IPMI) tables. The firmware setup programs are part of the discovery process at system power-up. The firmware is stored in system memory. The Basic Input/Output System (BIOS) performs initial queries after power-up. Thereafter, the operating system initializes itself and obtains information from industry standard tables such as SMBIOS and ACPI. Thus, the configuration proxy utilizes the same discovery mechanisms as the operation system in a pre-boot service environment.

[0024] The configuration proxy 330 takes advantage of the information gathering process administered by the EFI environment. Element 340 is a set of device drivers, which enable the communication between the configuration proxy and the platform firmware tables 390 by activating the firmware tables. The

device drivers 340 are unique depending on whether the architecture is based on IA-32 or IA-64.

[0025] When ready to run diagnostics, the test control 310, using the content of the test configuration files 320, invokes diagnostic providers 350. Diagnostic providers 350 are middleware interfaces that abstract test modules 370 from the test session control application 310. This middleware may be implemented as either an EFI application or an EFI hosted python script. The configuration information gathered by the configuration proxy 330 may be used to auto-configure the diagnostic test suites. The diagnostic provider 350 calls test modules 370 to execute the tests on hardware 380. For an embodiment of the invention, the test models 370 are responsible for initiating call backs to configuration proxy 330 in order to gather the appropriate method to access the device under test.

[0026] Further, the diagnostic provider 350 relies on Field Replaceable Units (FRU) isolators 360 to identify failed replacement units. The FRU isolators can utilize the configuration proxy 330 to access the SMBIOS structures in the platform tables 390 and other configuration information to identify failed FRU's.

[0027] Aside from invoking diagnostic providers 350, the test control 310 may also store information in system logs 325. The information stored in the system logs 325 can then be assessed by the diagnostic console 300 through an application programming interface provided by test control 310. In some

embodiments of the invention, the system logs may be accessed out of band through an apparatus such as a system management controller.

[0028] Figure 4 depicts a diagram of the configuration proxy 330. The configuration proxy 330 is a platform diagnostics component that is issued to dynamically determine the target system's device configuration and to provide an abstracted object interface to the platform's configuration information sources through a set of executables 410. For one embodiment, a separate set of diagnostic support components for each system. The configuration proxy 430 enables each system to have a common framework. The common framework enables diagnostic components to be used with more than one platform. This robust implementation can be achieved on EFI supported systems by compiling the configuration proxy executables 410 in the EFI environment.

[0029] In addition to supporting abstractions of sources of system configuration information, the configuration proxy 430 also provides an interface for managing the stored configuration data. The external software interface to the configuration proxy service is abstracted by an interface library 400. Each device table may be accessed through a specific group of objects and methods as defined by the specific device driver. For example, ACPI has a predefined set of objects and methods required to access ACPI information from the tables. Similarly, SMBIOS has a set of predefined objects and methods. The objects and methods of each firmware device conform to their respective industry standards.

[0030] The external software interface to the configuration proxy service is abstracted by the interface library 400. The configuration proxy 430 provides persistent storage 430 of the configuration data. The configuration proxy 330 relies on the content of the persistent storage file 430 to keep track of the previously detected configuration information retrieved from the ACPI tables and other platform configuration information sources.

[0031] The recently detected configuration is stored in working storage 420. A working storage 420 is maintained by the configuration proxy 330. The working data reflects the discovered configuration. The discovered data can be different from the expected data. Thus, the architecture identifies installed devices, or objects, that are not responsive by comparing the persistent data with the working data.

[0032] The persistent data, however, should be pre-configured because the persistent data is a snapshot of the expected configuration. Another operational mode such as reboot will detect if the configuration has changed. The pre-configuration is performed during an operational mode by putting the tables into file format and then downloading the file into the persistent storage 430. Moreover, the persistent data can be used to detect if the firmware is bad at machine bootup by comparing the expected configuration to the detected configuration.

[0033] In addition, the pre-configured data is also useful for certification purposes. For example, the platform configuration can be certified by

comparison with the pre-configured, persistent data. The persistent data is updated each time a change occurs. Therefore, there should be no changes that the persistent data does not know about.

[0034] Embodiments of the present invention may be implemented in hardware or software, or a combination of both. However, preferably, embodiments of the invention may be implemented in computer programs executing on programmable computer systems each comprising at least one processor, a data storage system (including volatile and non-volatile memory and/or storage elements), at least one input device, and at least one output device. Program code may be applied to input data to perform the functions described herein and generate output information. The output information may be applied to one or more output devices, in known fashion.

[0035] Each program may be implemented in a high level procedural or object oriented programming language to communicate with the computer system. However, the programs may be implemented in assembly or machine language, if desired. In any case, the language may be a compiled or interpreted language.

[0036] Each such computer program may be stored on a storage media or device (e.g., hard disk drive, floppy disk drive, read only memory (ROM), CD-ROM device, flash memory device, digital versatile disk (DVD), or other storage device) readable by a general or special purpose programmable computer system, for configuring and operating the computer system when the storage

media or device is read by the computer system to perform the procedures described herein. Embodiments of the invention may also be considered to be implemented as a machine-readable storage medium, configured for use with a computer system, where the storage medium so configured causes the computer system to operate in a specific and predefined manner to perform the functions described herein.

[0037] In the foregoing specification the invention has been described with reference to specific exemplary embodiments thereof. It will, however, be evident that various modification and changes may be made thereto without departure from the broader spirit and scope of the invention as set forth in the appended claims. The specification and drawings are, accordingly, to be regarded in an illustrative rather than restrictive sense.